Ant Colony Algorithm Applied to Automatic speech Recognition Graph Decoding
Benjamin Lecouteux, Didier Schwab

To cite this version:
Benjamin Lecouteux, Didier Schwab. Ant Colony Algorithm Applied to Automatic speech Recognition Graph Decoding. Interspeech, 2015, Dresden, Germany. hal-02094738

HAL Id: hal-02094738
https://hal.archives-ouvertes.fr/hal-02094738
Submitted on 9 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Ant Colony Algorithm Applied to Automatic speech Recognition Graph Decoding

Benjamin Lecouteux and Didier Schwab

LIG Université Grenoble-Alpes
GETALP Team

http://www.liglab.fr/
http://getalp.imag.fr

Introduction

We propose an original approach that allows the decoding of ASR Graphs by using a constructive algorithm based on ant colonies. In classical approaches, when a graph is decoded with higher order LM, the algorithm must expand the graph in order to develop each new observed n-gram. This extension process increases the computation time and memory consumption. We propose to use an ant colony algorithm in order to explore ASR graphs with a new language model, without the necessity of expanding it.

Related work

Classical approaches:
- **Synchronous graph algorithms**: generate a virtual copy of the graph for all hypotheses and that terminate after a fixed time.
- **Reentrant graph algorithms**: a virtual copy of the graph is explored for every linguistic context.
- **Stack asynchronous algorithms**: deep exploration and prioritize promising hypotheses.
- The rescoring of a 2-gram graph with 3-gram language model increases drastically the number of nodes.

In order to remove the language model expansion phase, we propose to use an ant colony algorithm to explore the graph.

ASR system, used corpora and baseline results

**ASR system based on the KALDI toolkit**
- Acoustic model trained on LIUM corpora: 118 hours of annotated data. Δ, ΔΔ coefficient parameters and LDA, MLLT are applied. SAT-fMLLR adaptation is used.
- LM trained on data provided for the IWSLT 2010 campaign as well as the training corpora provided by LIUM.

<table>
<thead>
<tr>
<th>LM order</th>
<th>n-grams</th>
<th>Perplexity</th>
<th>dev WER</th>
<th>test WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2g</td>
<td>35M</td>
<td>234</td>
<td>22.01%</td>
<td>21.67%</td>
</tr>
<tr>
<td>3g</td>
<td>232M</td>
<td>159</td>
<td>18.1%</td>
<td>17.7%</td>
</tr>
<tr>
<td>4g</td>
<td>524M</td>
<td>150</td>
<td>17.4%</td>
<td>16.7%</td>
</tr>
</tbody>
</table>

**TABLE:** Details on the different language models. (lexicon: 41K words)

**Used Corpora and baseline results**
- Experiments performed on the TED corpus 2010 that corresponds to a set of conference talks recorded in English.
- We purposely used an initial bi-gram language model in order to not introduce additional information at the linguistic level in the ASR.
- Then the graphs were rescored without pruning with 3 and 4-gram language models.

Proposed approaches

**ASR**: We use an initial bi-gram language model, so that we can use the ant colony algorithm to expand the graph to higher n-gram orders.

**Algorithm**: Each link of the graph is associated with a pheromone variable initialized at the beginning of each epoch. Our ants are launched from the first node of the graph and in order to leave a node, ants will have to choose a link. This choice is performed with a weighted random selection based on the quantity of pheromone on each arc. With the arrival of an ant at the endpoint, if a solution is found that is better than the previous best solution found, then pheromone is added on the path.

Experiments

<table>
<thead>
<tr>
<th>Set</th>
<th>Vit. no prune</th>
<th>Vit. 1xRT</th>
<th>1xRT</th>
<th>Ant 1 run</th>
<th>10 run</th>
<th>100 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev 3g</td>
<td>18.1</td>
<td>20.2</td>
<td>18.5</td>
<td>18.4</td>
<td>18.3</td>
<td></td>
</tr>
<tr>
<td>Dev 4g</td>
<td>17.4</td>
<td>19.1</td>
<td>17.8</td>
<td>17.7</td>
<td>17.7</td>
<td></td>
</tr>
<tr>
<td>Tst 3g</td>
<td>17.7</td>
<td>19.8</td>
<td>18.0</td>
<td>17.9</td>
<td>17.8</td>
<td></td>
</tr>
<tr>
<td>Tst 4g</td>
<td>16.7</td>
<td>18.5</td>
<td>17.2</td>
<td>17.0</td>
<td>17.0</td>
<td></td>
</tr>
<tr>
<td>Mem.3g 1 GB</td>
<td>18.1</td>
<td>20.2</td>
<td>18.5</td>
<td>18.4</td>
<td>18.3</td>
<td></td>
</tr>
<tr>
<td>Mem.3g 3 GB</td>
<td>17.4</td>
<td>19.1</td>
<td>17.8</td>
<td>17.7</td>
<td>17.7</td>
<td></td>
</tr>
<tr>
<td>Time3g 32h</td>
<td>17.7</td>
<td>19.8</td>
<td>18.0</td>
<td>17.9</td>
<td>17.8</td>
<td></td>
</tr>
<tr>
<td>Time4g 45h</td>
<td>16.7</td>
<td>18.5</td>
<td>17.2</td>
<td>17.0</td>
<td>17.0</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE:** We report the decoding time (dev,test), the memory footprint (average for one graph) and the WER compared to Viterbi beam-search baseline. We compare decoding time vs performance.

Discussion and Future Works

- Parallelization is trivial: ants can evolve independently of each other.
- Computational time is controlled.
- There are no physical extensions: memory usage is constant.
- The evaluation is performed when ant arrives at the final node.

- We wish to explore and analyze this ant colony paradigm more in depth.
- Use of ants with different behaviors.
- We would like to apply ant colony algorithms throughout the ASR process: from the phoneme lattice to the language model expansion.

**Figure**: Evolution of rescoring according the number of ant by node (on the dev set) during 1 run and comparison between Viterbi beam-search and ant colonies algorithm: Computing time against performance, on the dev set with a 4-gram language model. With time constraints the ant algorithm is more efficient than a conventional extension.